PRÁCTICA 3:

SISTEMA DE GESTIÓN PARA REFUGIO DE ANIMALES

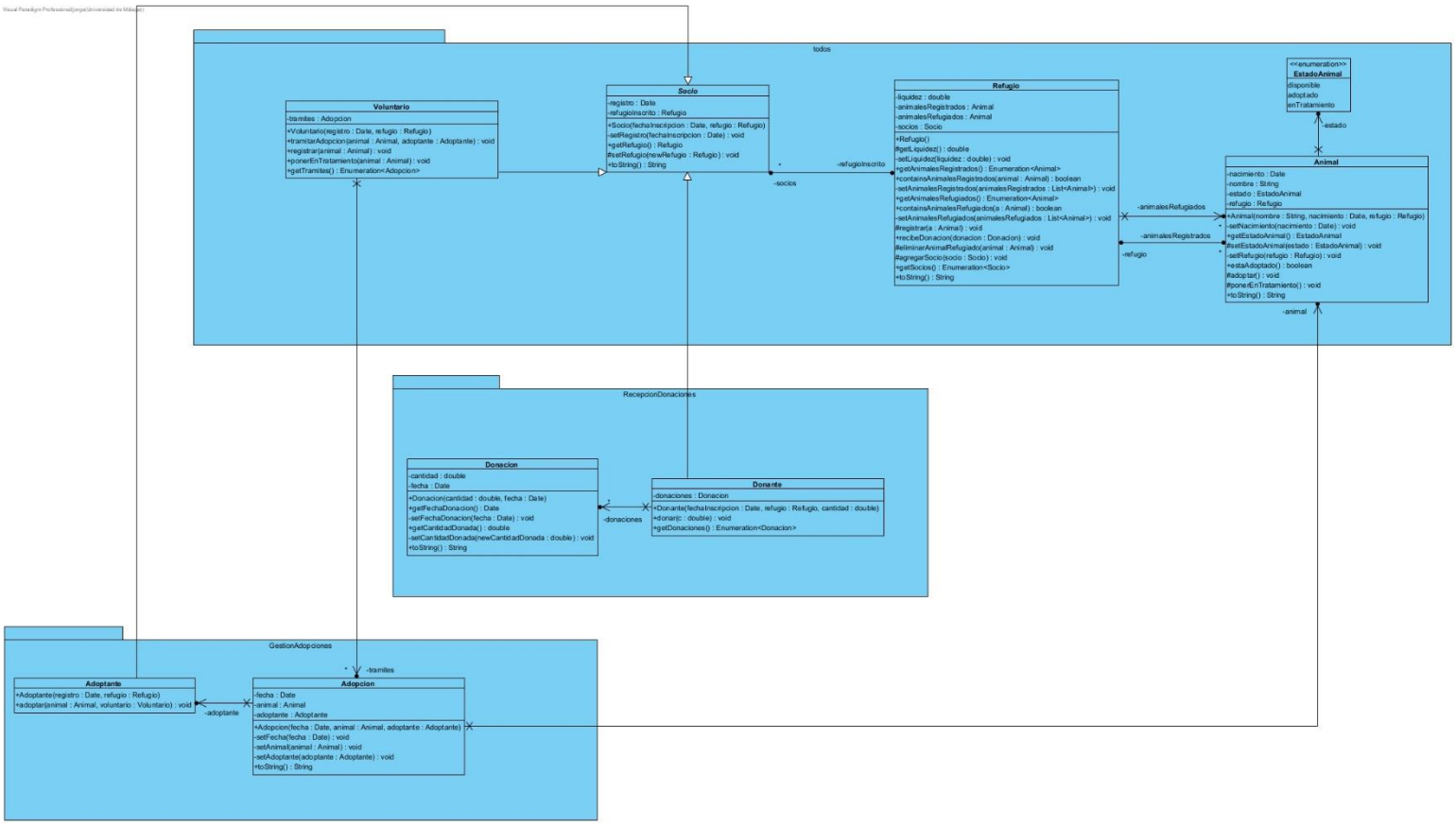
- Juan Manuel Valenzuela González – amcgil@uma.es
- Artur Vargas Carrión – arturvargas797@uma.es
- Jorge ReApullo Serrano – jorgers4@uma.es
- Eduardo González Bautista – edugb@uma.es
- Rubén Oliva Zamora – rubenoliva@uma.es
- David Muñoz del Valle – davidmunvalle@uma.es
- Alejandro Jiménez González – alejg411@uma.es

Modelado y Diseño del Software

GR1-3 | Ingeniería del Software A | Universidad de Málaga

ÍNDICE DE CONTENIDO

1. Andamiaje



1 Diagrama de diseño del apartado A del sistema de gestión para un refugio de animales en Visual Paradigm

1.1. Apartado A. Discusión

El diseño del **código de andamiaje** necesario para implementar el modelo UML se basa en reflejar directamente las clases, atributos y relaciones definidas en el diagrama. Cada clase, como `Animal`, `Adopcion` o `Refugio`, se traduce en una clase Java con sus correspondientes atributos y métodos. A continuación, se detalla más profundamente la estructura del código de andamiaje:

- **Uso de abstract class en Socio**: en el modelo UML, la clase `Socio` es declarada como abstracta para evitar que pueda ser instanciada directamente, ya que conceptualmente no tiene sentido tener un socio que no sea específicamente un `Adoptante`, un `Voluntario` o un `Donante`. En Java, esto se implementa mediante el uso de la palabra clave `abstract`, lo que asegura que Socio solo puede servir como una clase base para sus subclases. Esto permite definir las propiedades y comportamientos comunes a todos los tipos de socios, como la propiedad registro y la asociación con un Refugio. La clase abstracta Socio proporciona una base común que fomenta la reutilización de código y asegura consistencia en las características compartidas, además de facilitar la extensibilidad del sistema.

- **Uso de herencia**: las clases `Adoptante`, `Voluntario` y `Donante` heredan de `Socio` (usando `extends`) porque comparten características comunes, pero cada una representa un rol

específico dentro del sistema. Esta especialización permite agregar funcionalidades particulares a cada tipo de socio mientras se reutiliza el código definido en `Socio` (encapsulación de responsabilidades únicas de cada rol en sus respectivas clases). Por ejemplo:
    - Un `Adoptante` puede adoptar animales a través de un voluntario.
    - Un `Voluntario` tiene la capacidad de registrar animales y tramitar adopciones.
    - Un `Donante` puede realizar donaciones al refugio.

- **Relaciones entre clases:** las relaciones entre las clases se implementan utilizando colecciones dinámicas de objetos (por ejemplo, `List<Socio> socios` en `Refugio`) para gestionar extremos de cardinalidad 'muchos' u objetos para los extremos de asociaciones con cardinalidad 1 (por ejemplo, `Refugio refugioInscrito` en `Socio`).

- **Uso de métodos de acceso y modificación (getters/setters):** el uso de getters y setters garantiza el encapsulamiento, lo que significa que los atributos de las clases se acceden y modifican de manera controlada. Esto no solo asegura la validez de los datos, sino que también permite añadir lógica de validación o modificación en el futuro si es necesario, facilitando la mantenibilidad del código.

- **Uso de Enumeration en listas:** es importante que al hacer un get de algún atributo en formato lista de una clase, este no sea modificable. La forma de alcanzar este comportamiento es que los get sean definidos devolviendo un objeto de tipo Enumeration, en lugar de tipo List.

- **Visibilidad:** el proyecto se divide en tres paquetes para controlar el uso de métodos no permitidos entre clases (`protected`). Con ello también controlamos que no se usen los métodos que no deben de ser accesibles desde el main (solo son accesibles desde el main los métodos `public` porque el main no está en ningún paquete).

- **Control de errores:** combina asserts y excepciones para cubrir diferentes necesidades. Los asserts se utilizan durante el desarrollo para validar supuestos internos y evitar condiciones imposibles, como listas no inicializadas o relaciones mal definidas. Por otro lado, las excepciones manejan errores en tiempo de ejecución, como entradas inválidas o condiciones esperadas, garantizando la estabilidad del sistema y proporcionando mensajes claros al usuario. Este enfoque asegura robustez tanto en desarrollo como en producción.

1.2. Apartado A. Estrategias de implementación

La **estrategia de comportamiento condicional** ha sido elegida como base para implementar el código debido a la **simplicidad y claridad** que ofrece al manejar las transiciones y validaciones en un sistema con un número moderado de estados y relaciones. En el caso del sistema de gestión del refugio de animales, las operaciones como `adoptar` o `ponerEnTratamiento` tienen reglas claramente definidas y se gestionan eficientemente mediante validaciones condicionales (if-else) en los métodos correspondientes.

Otra decisión importante es la **no utilización de tablas de estados**. Aunque este enfoque podría centralizar las transiciones y proporcionar una representación más explícita de los estados y sus posibles transiciones, no se consideró necesario debido a la simplicidad de las reglas de transición. Implementar

tablas de estados sería excesivo para este sistema, dado que las transiciones entre estados son lineales y tienen poca interdependencia.

Finalmente, aunque el **patrón Estado** es una solución más avanzada y modular, su uso **tampoco fue necesario en este caso**. Este patrón es más adecuado para sistemas donde los estados son numerosos, dinámicos o donde los comportamientos asociados a los estados son complejos. En el sistema del refugio, los comportamientos específicos de cada estado son limitados y bien definidos, lo que hace que el uso de validaciones condicionales sea suficiente para manejar las reglas del dominio. La elección de no implementar el patrón Estado también ayuda a evitar la creación de clases adicionales y una posible sobrecarga en la estructura del código.

1.3. Apartado A. Main

Naturalmente, no vamos a copiar todo el código aquí. Por lo tanto, nos limitaremos a incluir el código main desarrollado y su salida esperada.

Main.java

```
import GestionAdopciones.Adoptante;
import RecepcionDonaciones.Donante;
import todos.Animal;
import todos.Refugio;
import todos.Voluntario;

import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;

public class Main {
    private static void imprimirRefugio(Refugio r) {
        String separador = "-----------------------------------";
        System.out.println();
        System.out.println(separador);
        System.out.println(r);
        System.out.println(separador);
        System.out.println();
    }

    private static <T> void imprimirEnumeration(Enumeration<T> enumeracion)
{
        while (enumeracion.hasMoreElements()) {
            T elemento = enumeracion.nextElement();
            System.out.println(elemento);
        }
    }

    public static void main(String[] args) {
        try {
            // Crear un refugio
            Refugio refugio = new Refugio();

            // Crear socios: varios voluntarios, adoptantes y donantes
            Voluntario voluntario1 = new Voluntario(new Date(), refugio);
            Voluntario voluntario2 = new Voluntario(new Date(), refugio);
            Adoptante adoptante1 = new Adoptante(new Date(), refugio);
            Adoptante adoptante2 = new Adoptante(new Date(), refugio);
```

```
            Donante donante1 = new Donante(new Date(), refugio, 5);
            Donante donante2 = new Donante(new Date(), refugio, 5000);

            imprimirRefugio(refugio);

            // Registrar múltiples animales
            Animal perro = new Animal("Coco", new Date(120,
Calendar.JANUARY, 1), refugio); // Fecha: 1 enero 2020
            Animal gato = new Animal("Michimini", new Date(121,
Calendar.JUNE, 10), refugio); // Fecha: 10 junio 2021
            Animal conejo = new Animal("Traviesito", new Date(119,
Calendar.MARCH, 15), refugio); // Fecha: 15 marzo 2019
            Animal loro = new Animal("Gor je", new Date(122,
Calendar.AUGUST, 5), refugio); // Fecha: 5 agosto 2022

            System.out.println("Registrando animales...");
            voluntario1.registrar(perro);
            voluntario1.registrar(gato);
            voluntario2.registrar(conejo);
            voluntario2.registrar(loro);
            System.out.println("Animales registrados con éxito.");

            imprimirRefugio(refugio);

            // Mostrar animales registrados en el refugio
            System.out.println("Animales registrados en el refugio:");
            imprimirEnumeration(refugio.getAnimalesRegistrados());
            System.out.println();

            // Adoptar varios animales
            System.out.println("Adoptando a Coco y a Michimini...");
            adoptante1.adoptar(perro, voluntario1);
            adoptante2.adoptar(gato, voluntario1);
            System.out.println("Adopciones tramitadas con éxito.");

            imprimirRefugio(refugio);

            // Mostrar animales refugiados tras las adopciones
            System.out.println("Animales refugiados después de las
adopciones:");
            imprimirEnumeration(refugio.getAnimalesRefugiados());
            System.out.println();

            // Realizar donaciones al refugio
            System.out.println("Realizando donaciones...");
            donante1.donar(100.0);
            donante2.donar(200.0);
            System.out.println("Donaciones realizadas con éxito.");

            imprimirRefugio(refugio);

            // Mostrar trámites realizados por los voluntarios
            System.out.println("Trámites realizados por los voluntarios:");
            System.out.println("todos.Voluntario 1:");
            imprimirEnumeration(voluntario1.getTramites());
            System.out.println("\ntodos.Voluntario 2:");
            imprimirEnumeration(voluntario2.getTramites());
            System.out.println();

            // Volver a poner un animal en tratamiento y registrar de nuevo
            System.out.println("Ponemos al conejo en tratamiento...");
```

```
            voluntario1.ponerEnTratamiento(conejo);
            System.out.println("Estado actual de Traviesito: " +
conejo.getEstadoAnimal());

            // Agregar más socios al refugio y mostrar el total
            System.out.println("Añadiendo 2 nuevos socios...");

            // Para comprobar que se meten al refugio correctamente
            Voluntario v2 = new Voluntario(new Date(), refugio);
            Adoptante a2 = new Adoptante(new Date(), refugio);


            imprimirEnumeration(refugio.getSocios());

            imprimirRefugio(refugio);

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

Salida

```
---------------------------------
todos.Refugio{liquidez=5005.0 euros, animalesRegistrados=0,
animalesRefugiados=0, socios=6}
---------------------------------

Registrando animales...
Animales registrados con éxito.

---------------------------------
todos.Refugio{liquidez=5005.0 euros, animalesRegistrados=4,
animalesRefugiados=4, socios=6}
---------------------------------

Animales registrados en el refugio:
todos.Animal{nombre='Coco', estado=disponible, nacimiento=01/01/2020
00:00:00}
todos.Animal{nombre='Michimini', estado=disponible, nacimiento=10/06/2021
00:00:00}
todos.Animal{nombre='Traviesito', estado=disponible, nacimiento=15/03/2019
00:00:00}
todos.Animal{nombre='Gor je', estado=disponible, nacimiento=05/08/2022
00:00:00}

Adoptando a Coco y a Michimini...
Adopciones tramitadas con éxito.

---------------------------------
todos.Refugio{liquidez=5005.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=6}
---------------------------------

Animales refugiados después de las adopciones:
todos.Animal{nombre='Traviesito', estado=disponible, nacimiento=15/03/2019
00:00:00}
todos.Animal{nombre='Gor je', estado=disponible, nacimiento=05/08/2022
00:00:00}
```

```
Realizando donaciones...
Donaciones realizadas con éxito.

---------------------------------
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=6}
---------------------------------

Trámites realizados por los voluntarios:
todos.Voluntario 1:
GestionAdopciones.Adopcion{fecha=02/12/2024 17:48:10,
animal=todos.Animal{nombre='Coco', estado=adoptado, nacimiento=01/01/2020
00:00:00}, adoptante=ADOPTANTE | Fecha de inscripción: 02/12/2024 17:48:10;
todos.Refugio: todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=6}}
GestionAdopciones.Adopcion{fecha=02/12/2024 17:48:10,
animal=todos.Animal{nombre='Michimini', estado=adoptado,
nacimiento=10/06/2021 00:00:00}, adoptante=ADOPTANTE | Fecha de
inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=6}}

todos.Voluntario 2:

Ponemos al conejo en tratamiento...
Estado actual de Traviesito: enTratamiento
Añadiendo 2 nuevos socios...
VOLUNTARIO | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
VOLUNTARIO | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
ADOPTANTE | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
ADOPTANTE | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
DONANTE | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
DONANTE | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
VOLUNTARIO | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
ADOPTANTE | Fecha de inscripción: 02/12/2024 17:48:10; todos.Refugio:
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}

---------------------------------
todos.Refugio{liquidez=5305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=8}
---------------------------------
```

2. Cambio en el diseño. Justificaciones y propuestas

2.1. Apartado B. ¿Por qué no vale lo que tenemos cuando un socio tiene más de un rol?

El problema de que un socio pueda tener más de un rol, como ser voluntario y adoptante al mismo tiempo, no está resuelto por las clases que se han creado en el código Java del apartado A debido a que **Java no soporta herencia múltiple**.

En el modelo actual, `Socio` es una clase abstracta que sirve como la base para otras clases concretas, como `Adoptante`, `Voluntario` y `Donante`. Si un `Socio` tiene múltiples roles, en el diseño actual no se contempla una manera flexible para asignar más de un rol a la misma persona. El uso de herencia única en Java implica que **una clase solo puede heredar de una clase base** (solo se permite poner un `extends`), lo que significa que un `Socio` no puede ser simultáneamente de más de un tipo, como un `Voluntario` y un `Adoptante`.

2.2. Apartado C. Propuestas de solución

Para permitir que un `Socio` tenga múltiples roles, podemos encontrar diversas soluciones:

- **Interfaces para roles:** podríamos crear interfaces como `Adoptante`, `Voluntario`, y `Donante`, y hacer que la clase `Socio` implemente los 3. De esta forma, cada instancia podría ejecutar los métodos asociados a cada uno de los roles. Para que una instancia no tenga la restricción de pertenecer a las 3 a la vez, se podrían añadir 3 booleanos privados a la clase, cada uno para un rol. Estos booleanos se solicitarían en el constructor, y se podrían obtener por medio de Getters y Setters. Si se llamara a algún método de un rol, y su booleano estuviese en false, elevaría una excepción. Esta implementación aseguraría que una clase `Socio` perteneciese desde 1 hasta todos los roles, con comprobación al ejecutarse los métodos, y asegurando poder añadir o eliminar roles sin necesidad de borrar el objeto.

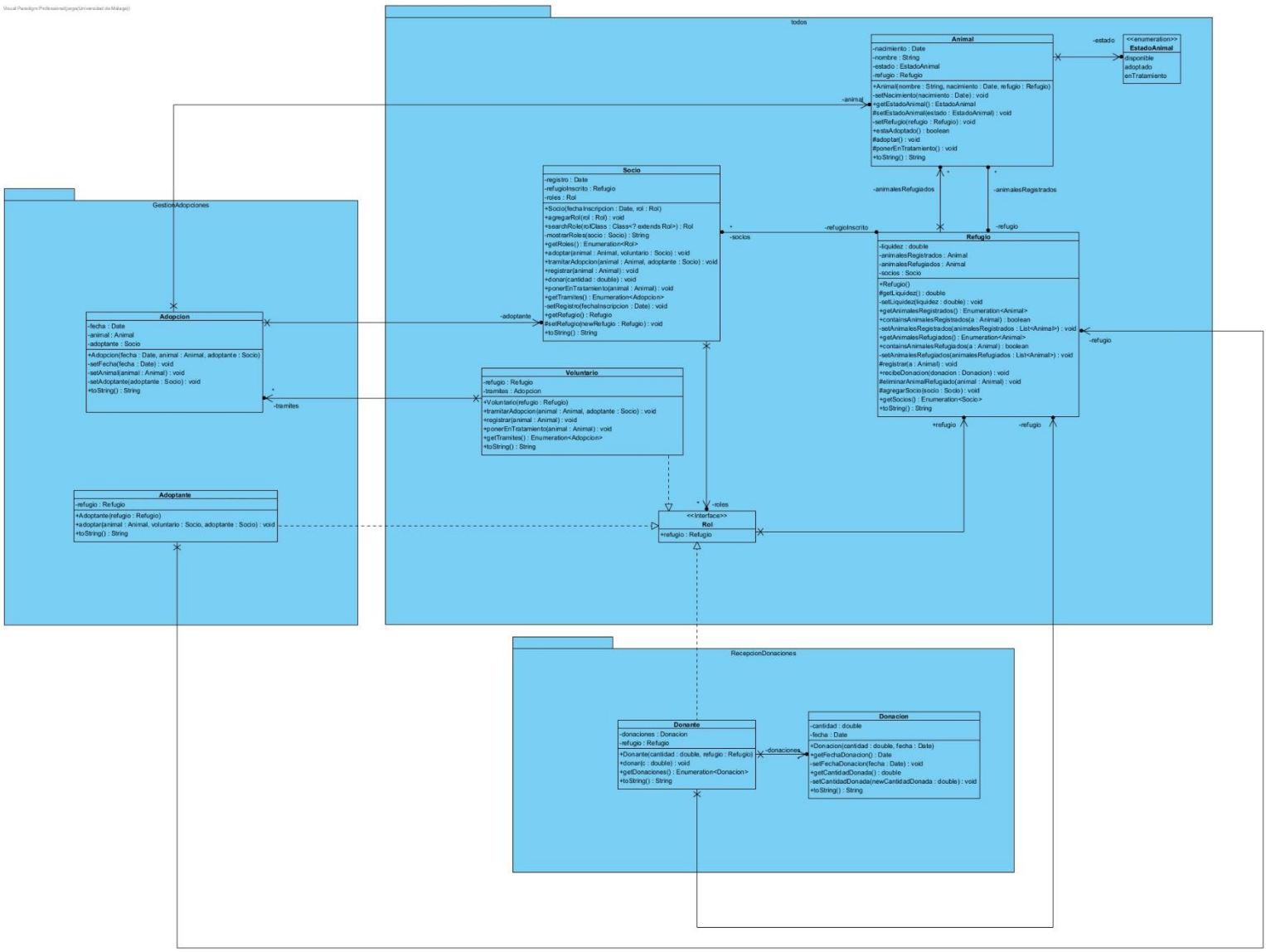
- **Composición de roles:** otra opción sería modelar los roles como objetos dentro de la clase `Socio`. En este caso, la clase `Socio` tendría una colección de objetos de tipo `Adoptante`, `Voluntario` y `Donante`, y un mismo `Socio` podría ser un `Voluntario`, `Adoptante` y `Donante` al mismo tiempo. Esto permite que el `Socio` tenga más de un rol de manera flexible.

- **Lista de enumerados:** se podría llegar a modelar, aunque la implementación acabaría siendo algo enrevesada. Para llevarla a cabo, se debería incluir en la clase `Socio` una lista de roles contenidos en un enumerado e incluir los métodos asociados a todos los roles dentro de la propia clase. Los métodos comprobarían antes de ejecutarse que el rol necesario para llevar a cabo la acción se encuentra en la lista, y elevarían una excepción en caso de que no se encontrara en su contenido.

- **Sistema de roles dinámicos:** en esta opción, implementaríamos una interfaz `Rol` que define métodos comunes para los diferentes roles. Los 3 roles que tenemos implementarían esta interfaz y añadirían sus métodos propios. La clase `Socio` tendrá una lista de roles que se irá actualizando en tiempo de ejecución. De esta forma podremos añadir los roles asociados al socio, e incluso crear roles nuevos, sin tocar los ya existentes. Para ejecutar una acción, se buscaría el rol correspondiente en la lista y se ejecutaría su método. Si este rol no está, se elevaría una excepción.

Tras valorar las posibles soluciones hemos decidido implementar la de **sistema de roles dinámicos**. Esta implementación nos permite añadir nuevos roles solamente añadiendo un método proxy a la clase `Socio`, simplemente creando una nueva implementación de la interfaz `Rol`. Además de ser tan flexible, aseguramos el cumplimiento de los principios SOLID, garantizando un diseño modular y extensible. Por ejemplo, el principio de abierto/cerrado (OCP) se respeta al permitir añadir nuevos roles simplemente implementando la interfaz `Rol`, sin modificar el código existente. Asimismo, el principio de responsabilidad única (SRP) se cumple al encapsular la lógica específica de cada rol en su propia clase.

3. Implementación del sistema de roles dinámicos



2 Diagrama de diseño del apartado D del sistema de gestión para un refugio de animales en Visual Paradigm

3.1. Apartado D. Aclaraciones

El código en Java ha planteado ciertos desafíos en términos de optimización y viabilidad, lo que ha llevado a realizar diversas pruebas para identificar la solución más eficiente. El cambio más relevante se ha introducido en la clase `Socio`, donde se ha implementado un enfoque **Proxy**. Este patrón actúa como intermediario para los métodos asociados a los roles de `Adoptante`, `Donante` y `Voluntario`, permitiendo que la clase `Socio` filtre y distribuya las solicitudes según el rol correspondiente.

Además, se han eliminado los métodos `super()` de los roles, ya que estos ya no forman parte de una relación de herencia con la clase `Socio`, sino que ahora son invocados directamente a través de esta última. Este rediseño simplifica la estructura y mejora la flexibilidad en la interacción entre la clase

`Socio` y sus roles. Respecto a los aspectos más importantes tomados en cuenta a la hora de implementar la solución, destacan los siguientes:

- **Interfaz `Rol`:** adoptado un sistema de roles dinámicos basado en la implementación de una interfaz común llamada `Rol`, que define los métodos compartidos entre los diferentes roles. Los roles de `Adoptante`, `Voluntario` y `Donante` implementan esta interfaz, añadiendo sus métodos específicos. La clase `Socio` gestiona una lista dinámica de roles que puede actualizarse en tiempo de ejecución, permitiendo asignar nuevos roles o crear otros adicionales sin modificar los existentes. Para ejecutar una acción, se busca el rol correspondiente en la lista y se invoca su método; si el rol no está presente, se eleva una excepción.

- **Eliminación del modificador abstract:** en la clase `Socio` dejando de ser abstracta y pudiendo ser instanciada, permitiendo gestionar instancias directamente desde esta clase, en lugar de depender únicamente de subclases concretas.

- **Implementación de métodos Proxy para roles:** se añaden métodos específicos que delegan tareas a los roles correspondientes:
  - `tramitarAdopcion`: delega en un rol de tipo `Voluntario`.
  - `Registrar`: delega en un rol de tipo `Voluntario`.
  - `Adoptar`: combina roles de `Adoptante` y `Voluntario` para gestionar la adopción.
  - `Donar`: delega en un rol de tipo `Donante`.
  - `getTramites`: delega en un rol de tipo `Voluntario`.
  - `ponerEnTratamiento`: delega en un rol de tipo `Voluntario`.

- **Cambio en el constructor de Socio:** se recibe una lista con elementos de la clase `Rol` como parámetro y usándolo para inicializar la lista de roles. Esta lista no podrá ser ni nula ni vacía ya que los socios deben siempre poder desempeñar mínimo un rol. En otro caso elevará una excepción.

- **Validaciones y excepciones relacionadas con roles**: en esta implementación cada `Socio` podrá implementar unas determinadas acciones según los roles que tenga determinados, por tanto, se deberá verificar que el `Socio` tenga los roles requeridos antes de ejecutar cualquier acción. Si estos no están presentes, se elevará una excepción.

  Ningún socio podrá tener dos veces el mismo rol. Para evitar esto, en la clase socio cada vez que se añada un rol al socio, se comprobará que este no pertenezca ya a la lista de roles.

Con respecto a los nuevos métodos implementados para poder llevar a cabo el sistema de roles dinámicos, destacan los siguientes:

- Método `searchRole`: encapsula la lógica de búsqueda de un rol específico que está asociado a un socio, además de proporcionarnos una forma de verificar si un `Socio` tiene un rol particular sin recorrer al completo la lista de roles lo que simplifica la validación de los roles a la hora de usarlo en otras funciones.

```
public Rol searchRole(Class<? extends Rol> rolClass) {
    for (Rol rol : roles) {
       if (rolClass.isInstance(rol)) {
          return rol;
       }
    }
    return null;
}
```

- Método `agregarRol`: tiene como objetivo añadir roles adicionales a un Socio de manera dinámica ofreciéndonos así una asignación controlada y coherente de roles.

```
public void agregarRol(Rol rol) {
    if (refugioInscrito != rol.getRefugio()) {
       throw new IllegalArgumentException("Todos los roles tienen que estar
en el mismo refugio.");
    }
    if (roles.toString().contains(rol.toString())) {
       System.err.printf("El rol %s ya estaba asignado al socio. No se ha
introducido de nuevo.%n", rol);
    } else {
       roles.add(rol);
    }
}
```

- Método `mostrarRoles`: genera una representación textual de los roles asignados a un `Socio`, necesario para poder mostrar mensajes correctos en el control de errores.

```
private String mostrarRoles(Socio socio) {
    StringJoiner sj = new StringJoiner(";");
    Enumeration<Rol> roles = socio.getRoles();
    while( roles.hasMoreElements() ) {
       sj.add( roles.nextElement().toString() );
    }
    return sj.toString();
}
```

- Método `adoptar`: método proxy que da permisos para adoptar a un animal a todo aquel que tenga el rol de `Adoptante` y a su vez valida que el proceso esté supervisado por un socio con el rol de `Voluntario`.

```
    public void adoptar(Animal animal, Socio voluntario){
        Rol rolAdoptante = searchRole(Adoptante.class);
        if (rolAdoptante == null) {
            throw new IllegalArgumentException("Este socio no tiene el rol de
GestionAdopciones.Adoptante: " + roles.toString());
        }
        Rol rolVoluntario = voluntario.searchRole(Voluntario.class);
        if (rolVoluntario == null) {
            throw new IllegalArgumentException("Este socio no tiene el rol de
todos.Voluntario: " + mostrarRoles(voluntario) );
        }
        ((Adoptante) rolAdoptante).adoptar(animal, voluntario, this);
    }
```

Los demás métodos proxy **funcionan exactamente igual** que adoptar. Se busca el rol, si no se encuentra se eleva una excepción y si sí se encuentra, **se ejecuta el método de igual nombre contenido en la clase del rol que corresponda**. Para más información se ruega consultar el código entregado.

3.2. Apartado D. Main

Al igual que en el apartado A, no vamos a copiar todo el código aquí. De igual forma, copiaremos el main del apartado D (muy similar al del apartado A) y mostraremos su salida esperada.

Main.java

```
import GestionAdopciones.Adoptante;
import RecepcionDonaciones.Donante;
import todos.Animal;
import todos.Refugio;
import todos.Socio;
import todos.Voluntario;

import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;

public class Main {
    private static void imprimirRefugio(Refugio r) {
        String separador = "----------------------------------";
        System.out.println();
        System.out.println(separador);
        System.out.println(r);
        System.out.println(separador);
        System.out.println();
    }

    private static <T> void imprimirEnumeration(Enumeration<T> enumeracion)
{
        while (enumeracion.hasMoreElements()) {
            T elemento = enumeracion.nextElement();
            System.out.println(elemento);
        }
    }

    public static void main(String[] args) {
        try {
            // Crear un refugio
            Refugio refugio = new Refugio();
            //Refugio refugio2 = new Refugio();

            // Crear socios: varios voluntarios, adoptantes y donantes

            Socio voluntarioDonante = new Socio(new Date(),new
Voluntario(refugio));
            Socio voluntarioAdoptante = new Socio(new Date(),new
Voluntario(refugio));

            voluntarioDonante.agregarRol(new Donante(5, refugio));
            voluntarioAdoptante.agregarRol(new Adoptante(refugio));

            imprimirRefugio(refugio);

            // Registrar múltiples animales
            Animal perro = new Animal("Coco", new Date(120,
Calendar.JANUARY, 1), refugio); // Fecha: 1 enero 2020
            Animal gato = new Animal("Michimini", new Date(121,
Calendar.JUNE, 10), refugio); // Fecha: 10 junio 2021
            Animal conejo = new Animal("Traviesito", new Date(119,
```

```
Calendar.MARCH, 15), refugio); // Fecha: 15 marzo 2019
            Animal loro = new Animal("Gor je", new Date(122,
Calendar.AUGUST, 5), refugio); // Fecha: 5 agosto 2022

            System.out.println("Registrando animales...");
            voluntarioDonante.registrar(perro);
            voluntarioDonante.registrar(gato);
            voluntarioAdoptante.registrar(conejo);
            voluntarioAdoptante.registrar(loro);
            System.out.println("Animales registrados con éxito.");

            imprimirRefugio(refugio);

            // Mostrar animales registrados en el refugio
            System.out.println("Animales registrados en el refugio:");
            imprimirEnumeration(refugio.getAnimalesRegistrados());
            System.out.println();

            // Adoptar varios animales
            System.out.println("Adoptando a Coco y a Michimini...");
            voluntarioAdoptante.adoptar(perro, voluntarioDonante);
            voluntarioAdoptante.adoptar(gato, voluntarioDonante);
            System.out.println("Adopciones tramitadas con éxito.");

            imprimirRefugio(refugio);

            // Mostrar animales refugiados tras las adopciones
            System.out.println("Animales refugiados después de las
adopciones:");
            imprimirEnumeration(refugio.getAnimalesRefugiados());
            System.out.println();

            // Realizar donaciones al refugio
            System.out.println("Realizando donaciones...");
            voluntarioDonante.donar(100.0);
            voluntarioDonante.donar(200.0);
            System.out.println("Donaciones realizadas con éxito.");

            imprimirRefugio(refugio);

            // Mostrar trámites realizados por los voluntarios
            System.out.println("Trámites realizados por los voluntarios:");
            System.out.println("todos.Voluntario 1:");
            imprimirEnumeration(voluntarioDonante.getTramites());
            System.out.println("\ntodos.Voluntario 2:");
            imprimirEnumeration(voluntarioDonante.getTramites());
            System.out.println();

            // Volver a poner un animal en tratamiento y registrar de nuevo
            System.out.println("Ponemos al conejo en tratamiento...");
            voluntarioAdoptante.ponerEnTratamiento(conejo);
            System.out.println("Estado actual de Traviesito: " +
conejo.getEstadoAnimal());

            // Agregar más socios al refugio y mostrar el total
            System.out.println("Añadiendo 2 nuevos socios...");

            // Se crean para comprobar que se añaden al refugio
            Socio v2s = new Socio(new Date(), new Voluntario(refugio));
            Socio a2s = new Socio(new Date(), new Adoptante(refugio));
```

```
            imprimirEnumeration(refugio.getSocios());

            imprimirRefugio(refugio);

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

Salida

```
---------------------------------
todos.Refugio{liquidez=5.0 euros, animalesRegistrados=0,
animalesRefugiados=0, socios=2}
---------------------------------

Registrando animales...
Animales registrados con éxito.

---------------------------------
todos.Refugio{liquidez=5.0 euros, animalesRegistrados=4,
animalesRefugiados=4, socios=2}
---------------------------------

Animales registrados en el refugio:
todos.Animal{nombre='Coco', estado=disponible, nacimiento=01/01/2020
00:00:00}
todos.Animal{nombre='Michimini', estado=disponible, nacimiento=10/06/2021
00:00:00}
todos.Animal{nombre='Traviesito', estado=disponible, nacimiento=15/03/2019
00:00:00}
todos.Animal{nombre='Gor je', estado=disponible, nacimiento=05/08/2022
00:00:00}

Adoptando a Coco y a Michimini...
Adopciones tramitadas con éxito.

---------------------------------
todos.Refugio{liquidez=5.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=2}
---------------------------------

Animales refugiados después de las adopciones:
todos.Animal{nombre='Traviesito', estado=disponible, nacimiento=15/03/2019
00:00:00}
todos.Animal{nombre='Gor je', estado=disponible, nacimiento=05/08/2022
00:00:00}

Realizando donaciones...
Donaciones realizadas con éxito.

---------------------------------
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=2}
---------------------------------

Trámites realizados por los voluntarios:
```

```
todos.Voluntario 1:
GestionAdopciones.Adopcion{fecha=02/12/2024 17:43:45,
animal=todos.Animal{nombre='Coco', estado=adoptado, nacimiento=01/01/2020
00:00:00}, adoptante=Socio | Fecha de inscripción: 02/12/2024 17:43:45;
todos.Refugio: todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=2}; Roles: [Voluntario, Adoptante]}
GestionAdopciones.Adopcion{fecha=02/12/2024 17:43:45,
animal=todos.Animal{nombre='Michimini', estado=adoptado,
nacimiento=10/06/2021 00:00:00}, adoptante=Socio | Fecha de inscripción:
02/12/2024 17:43:45; todos.Refugio: todos.Refugio{liquidez=305.0 euros,
animalesRegistrados=4, animalesRefugiados=2, socios=2}; Roles: [Voluntario,
Adoptante]}

todos.Voluntario 2:
GestionAdopciones.Adopcion{fecha=02/12/2024 17:43:45,
animal=todos.Animal{nombre='Coco', estado=adoptado, nacimiento=01/01/2020
00:00:00}, adoptante=Socio | Fecha de inscripción: 02/12/2024 17:43:45;
todos.Refugio: todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=2}; Roles: [Voluntario, Adoptante]}
GestionAdopciones.Adopcion{fecha=02/12/2024 17:43:45,
animal=todos.Animal{nombre='Michimini', estado=adoptado,
nacimiento=10/06/2021 00:00:00}, adoptante=Socio | Fecha de inscripción:
02/12/2024 17:43:45; todos.Refugio: todos.Refugio{liquidez=305.0 euros,
animalesRegistrados=4, animalesRefugiados=2, socios=2}; Roles: [Voluntario,
Adoptante]}

Ponemos al conejo en tratamiento...
Estado actual de Traviesito: enTratamiento
Añadiendo 2 nuevos socios...
Socio | Fecha de inscripción: 02/12/2024 17:43:45; todos.Refugio:
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=4}; Roles: [Voluntario, Donante]
Socio | Fecha de inscripción: 02/12/2024 17:43:45; todos.Refugio:
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=4}; Roles: [Voluntario, Adoptante]
Socio | Fecha de inscripción: 02/12/2024 17:43:46; todos.Refugio:
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=4}; Roles: [Voluntario]
Socio | Fecha de inscripción: 02/12/2024 17:43:46; todos.Refugio:
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=4}; Roles: [Adoptante]

---------------------------------
todos.Refugio{liquidez=305.0 euros, animalesRegistrados=4,
animalesRefugiados=2, socios=4}
---------------------------------
```